**UNIT- 3**

**LISTS, DICTIONARIES, FUNCTIONS AND MODULES**

**List and Dictionaries:** Lists, Defining Simple Functions, Dictionaries.

**Design with Function:** Functions as Abstraction Mechanisms, Problem Solving with Top Down Design, Design with Recursive Functions, Case Study Gathering Information from a File System, Managing a Program’s Namespace, Higher Order Function.

**Modules:** Modules, Standard Modules, Packages

**Lists:**

🡪 List is used to store the group of values & we can manipulate them, in list the values are stores in index format starts with 0.

🡪 List is mutable object so we can do the manipulations.

🡪A python list is enclosed between square ([]) brackets.

🡪In list insertion order is preserved it means in which order we inserted element same order

output is printed.

🡪 A list can be composed by storing a sequence of different type of values separated by commas.

🡪The list contains forward indexing & backward indexing.

**Syntax:** <list\_name> = [value1,value2,value3,...,valuen]

**Example:**

data1=[1,2,3,4] # list of integers

data2=['x','y','z'] # list of String

data3=[12.5,11.6] # list of floats

data4=[] # empty list

data5=['TEC',10,56.4,'a'] # list with mixed data types

**Properties of a list:**

🡪Mutable: The elements of the list can be modified. We can add or remove items to the list after it has been created.

🡪Ordered: The items in the lists are ordered. Each item has a unique index value. The new items

will be added to the end of the list.

🡪Heterogeneous: The list can contain different kinds of elements i.e; they can contain elements of string, integer, Boolean, or any type.

🡪Duplicates: The list can contain duplicates i.e., lists can have two items with the same values.

**Why use a list?**

🡪The list data structure is very flexible It has many unique inbuilt functionalities like pop(), append(), etc which makes it easier, where the data keeps changing.

🡪Also, the list can contain duplicate elements i.e two or more items can have the same values.

🡪 Lists are Heterogeneous i.e., different kinds of objects/elements can be added

🡪As Lists are mutable it is used in applications where the values of the items change frequently.

**Creating a Python list:**

The list can be created using either the list constructor or using square brackets [].

🡪Using list() constructor: In general, the constructor of a class has its class name. Similarly, Create a list by passing the comma-separated values inside the list().

🡪Using square bracket ([]): In this method, we can create a list simply by enclosing the items inside the square brackets.

my\_list1 = list((1, 2, 3))

print(my\_list1)

**# Output** [1, 2, 3]

# Using square brackets[]

my\_list2 = [1, 2, 3]

print(my\_list2)

**# Output** [1, 2, 3]

# with heterogeneous items

my\_list3 = [1.0, 'Jessa', 3]

print(my\_list3)

# Output [1.0, 'Jessa', 3]

# empty list using list()

my\_list4 = list()

print(my\_list4)

# Output []

# empty list using []

my\_list5 = []

print(my\_list4)

# Output []

**Length of a List:**

In order to find the number of items present in a list, we can use the len() function.

my\_list = [1, 2, 3]

print(len(my\_list))

**# output** 3

**Accessing items of a List:**

The items in a list can be accessed through indexing and slicing. This section will guide you by accessing the list using the following two ways

 Using indexing, we can access any item from a list using its index number

 Using slicing, we can access a range of items from a list

**Indexing**

The list elements can be accessed using the “indexing” technique. Lists are ordered collections with unique indexes for each item. We can access the items in the list using this index number.

**Python Positive and Negative indexing**

To access the elements in the list from left to right, the index value starts from zero to (length of the list-1) can be used. For example, if we want to access the 3rd element we need to use 2 since the index value starts from 0.

**Note:**

 As Lists are ordered sequences of items, the index values start from 0 to the Lists length.

 Whenever we try to access an item with an index more than the Lists length, it will throw the 'Index Error'.

 Similarly, the index values are always an integer. If we give any other type, then it will throw Type Error.

**Example**

my\_list = [10, 20, 'Jessa', 12.50, 'Emma']

# accessing 2nd element of the list

print(my\_list[1]) # 20

# accessing 5th element of the list

print(my\_list[4]) # 'Emma'

**Negative Indexing**

The elements in the list can be accessed from right to left by using negative indexing. The negative value starts from -1 to -length of the list. It indicates that the list is indexed from the reverse/backward.

my\_list = [10, 20, 'Jessa', 12.50, 'Emma']

# accessing last element of the list

print(my\_list[-1])

# output 'Emma'

# accessing second last element of the list

print(my\_list[-2])

# output 12.5

# accessing 4th element from last

print(my\_list[-4])

# output 20

**List Slicing**

Slicing a list implies, accessing a range of elements in a list. For example, if we want to get the elements in the position from 3 to 7, we can use the slicing method. We can even modify the values in a range by using this slicing technique.

The below is the syntax for list slicing.

listname[start\_index : end\_index : step]

 The start\_index denotes the index position from where the slicing should begin and

the end\_index parameter denotes the index positions till which the slicing should be done.

 The step allows you to take each nth-element within a start\_index:end\_index range.

**Example**

my\_list = [10, 20, 'Jessa', 12.50, 'Emma', 25, 50]

# Extracting a portion of the list from 2nd till 5th element

print(my\_list[2:5])

# Output ['Jessa', 12.5, 'Emma']

Let us see few more examples of slicing a list such as

 Extract a portion of the list

 Reverse a list

 Slicing with a step

 Slice without specifying start or end position

Example

my\_list = [5, 8, 'Tom', 7.50, 'Emma']

# slice first four items

print(my\_list[:4])

# Output [5, 8, 'Tom', 7.5]

# print every second element

# with a skip count 2

print(my\_list[::2])

# Output [5, 'Tom', 'Emma']

# reversing the list

print(my\_list[::-1])

# Output ['Emma', 7.5, 'Tom', 8, 5]

# Without end\_value

# Stating from 3nd item to last item

print(my\_list[3:])

# Output [7.5, 'Emma']

**Iterating a List**

The objects in the list can be iterated over one by one, by using a for a loop.

my\_list = [5, 8, 'Tom', 7.50, 'Emma']

# iterate a list

for item in my\_list:

print(item)

**Iterate along with an index number**

The index value starts from 0 to (length of the list-1). Hence using the function range() is ideal for this scenario. The range function returns a sequence of numbers. By default, it returns starting from 0 to the specified number (increments by 1). The starting and ending values can be passed according to our needs.

**Example:**

my\_list = [5, 8, 'Tom', 7.50, 'Emma']

# iterate a list

for i in range(0, len(my\_list)):

# print each item using index number

print(my\_list[i])

**Python List/Array Methods:** Python has a set of built-in methods that you can use on lists/arrays.

**Method Description**

**append():** Adds an element at the end of the list

**clear() :**Removes all the elements from the list

**copy()** :Returns a copy of the list

**count():** Returns the number of elements with the specified value

**extend():**Add the elements of a list (or any iterable), to the end of the current list

**index():** Returns the index of the first element with the specified value

**insert():**Adds an element at the specified position pop()

Removes the element at the specified position

**remove()** Removes the first item with the specified value

**reverse()** Reverses the order of the list

**sort()** Sortsthe list

**Note:** Python does not have built-in support for Arrays, but Python Lists can be used instead.

**1. List append():**

 The append() method appends an element to the end of the list.

**Syntax:** list.append(elmnt)

Here, elmnt Required. An element of any type (string, number, object etc.)

**Example: 1** Add an element to the fruits list.

fruits = ['apple', 'banana', 'cherry']

fruits.append("orange")

Output: ['apple', 'banana', 'cherry', 'orange']

**Example: 2** Add a list to a list.

a = ["apple", "banana", "cherry"]

b = ["Ford", "BMW", "Volvo"]

a.append(b)

Output: ['apple', 'banana', 'cherry', ["Ford", "BMW", "Volvo"]]

**2. List clear():**

 The clear() method removes all the elements from a list.

Syntax: list.clear()

**Example:** Remove all elements from the fruits list.

fruits = ['apple', 'banana', 'cherry', 'orange']

fruits.clear()

Output: []

**3. List copy():**

 The copy() method returns a copy of the specified list.

Syntax: list.copy()

**Example:** Copy the fruits list.

fruits = ['apple', 'banana', 'cherry', 'orange']

x =fruits.copy()

Output: ['apple', 'banana', 'cherry']

**4. List count()**

 The count() method returns the number of elements with the specified value.

Syntax: list.count(value)

Here,

value = Required. Any type (string, number, list, tuple, etc.). The value to search for.

**Example-1:** Return the number of times the value "cherry" appears in the fruits list.

fruits =['apple', 'banana', 'cherry']

x = fruits.count("cherry")

**Output:** 1

**Example-2:** Return the number of times the value 9 appears in the list.

points = [1, 4, 2, 9, 7, 8, 9, 3, 1]

x = points.count(9)

**Output:** 2

**5. List extend():** The extend() method adds the specified list elements (or any iterable) to the end of the current list.

Syntax: list.extend(iterable)

Here,

iterable Required. Any iterable (list, set, tuple, etc.)

**Example – 1:** Add the elements of cars to the fruits list.

fruits = ['apple', 'banana', 'cherry']

cars = ['Ford', 'BMW', 'Volvo']

fruits.extend(cars)

Output: ['apple', 'banana', 'cherry', 'Ford', 'BMW', 'Volvo']

**Example – 2:** Add a tuple to the fruits list.

fruits = ['apple', 'banana', 'cherry']

points = (1, 4, 5, 9)

fruits.extend(points)

**Output:** ['apple', 'banana', 'cherry', 1, 4, 5, 9]

**6. List index():** The index() method returns the position at the first occurrence of the specified value.

Note: The index() method only returns the first occurrence of the value.

Syntax: list.index(elmnt)

Here,

elmnt = Required. Any type (string, number, list, etc.). The element to search for.

Example – 1: What is the position of the value "cherry".

fruits = ['apple', 'banana', 'cherry']

x = fruits.index("cherry")

**Output:** 2

Example – 2: What is the position of the value 32.

fruits = [4, 55, 64, 32, 16, 32]

x = fruits.index(32)

**Output:** 3

**7. List insert():**

 To insert a new list item, without replacing any of the existing values, we can use the insert()

method. The insert() method inserts an item at the specified index:

Syntax: list.insert(pos, elmnt)

Here,

pos = Required. A number specifying in which position to insert the value

elmnt=Required. An element of any type (string, number, object etc.)

Example: Insert "watermelon" as the third item.

x= ["apple", "banana", "cherry"]

x.insert(2, "watermelon")

print(x)

**Output:** ['apple', 'banana', 'watermelon', 'cherry']

**8. List pop():**

 The pop() method removes the element at the specified position.

Note: The pop() method returns removed value.

Syntax: list.pop(pos)

Here,

Pos = Optional. A number specifying the position of the element you want to remove, default value is -1, which returns the last item

**Example - 1:** Remove the second element of the fruit list:

fruits = ['apple', 'banana', 'cherry']

fruits.pop(1)

Output: ['apple', 'cherry']

**Example – 2:** Return the removed element:

fruits = ['apple', 'banana', 'cherry']

x= fruits.pop(1)

**Output:** banana

**9. List remove() :**

 The remove() method removes the first occurrence of the element with the specified value.

Syntax: list.remove(elmnt)

Here,

element Required. Any type (string, number, list etc.) The element you want to remove

Example: Remove the "banana" element of the fruit list.

fruits = ['apple', 'banana', 'cherry']

fruits.remove("banana")

Output: ['apple', 'cherry']

**10. List reverse():**

 The reverse() method reverses the sorting order of the elements.

Syntax: list.reverse()

**Example:1** Reverse the order of the fruit list.

fruits= ['apple', 'banana', 'cherry']

fruits.reverse()

Output: ['cherry', 'banana', 'apple']

**Example:2**

List1=["banana", "Orange", "Kiwi", "cherry"]

List1.reverse()

print(List1)

**Output:** ['cherry', 'Kiwi', 'Orange', 'banana']

**Loop Through a List:**

 We can loop through the list items by using a for loop.

 Print all items in the list, one by one.

 Use the range() and len() functions to print all items by referring to their index number:

Example –1:

thislist = ["apple", "banana","cherry"]

for x in thislist:

print(x)

Output:

apple

banana

cherry

Example-2:

thislist = ["apple", "banana","cherry"]

for i in range(len(thislist)):

print(thislist[i])

Output:

apple

banana

cherry

 Using a While Loop: Use the len() function to determine the length of the list, then start at 0

and loop your way through the list items by refering to their indexes. Remember to increase the

index by 1 after each iteration.

Example – 1:

thislist = ["apple", "banana", "cherry"]

i =0

while i < len(thislist):

print(thislist[i])

i = i + 1

Output:

apple

banana

cherry

**Looping Using List Comprehensive:** It offers the shortest syntax for looping through lists.

Example – 1:

thislist = ["apple", "banana", "cherry"]

x=[ x for x in thislist]

Print(x)

Output:

Apple

banana

cherry

**Sort Lists:** Sort List Alphanumerically. List objects have a sort() method that will sort the list

alphanumerically, ascending, by default. To sort descending, use the keyword argument

reverse = True.

Example -1:

List1= ["orange", "mango", "kiwi", "pineapple", "banana"]

List1.sort()

print(List1)

Output: ['banana', 'kiwi', 'mango', 'orange', 'pineapple']

Example – 2:

List1 = [100, 50, 65, 82, 23]

List1.sort()

print(List1)

Output: [23, 50, 65, 82, 100]

Example – 3: Sort the list descending.

List1= ["orange", "mango", "kiwi" ,"pineapple", "banana"]

List1.sort(reverse =True)

print(List1)

Output: ['pineapple', 'orange', 'mango', 'kiwi', 'banana']

**Customize Sort Function:** we can also customize you own function by using the keyword argument key = function. The function will return a number that will be used to sort the list (the lowest number first):

Example -1: Sort the list based on how close the number is to 50:

def myfunc(n):

return abs(n - 50)

thislist = [100, 50, 65, 82, 23]

thislist.sort(key = myfunc)

print(thislist)

Output:

[50, 65, 23, 82, 100]

Note: By default the sort() method is case sensitive, resulting in all capital letters being sorted after lower case letters:

Example – 1:

List1 = ["banana", "Orange", "Kiwi", "cherry"]

List1.sort()

print(List1)

**Output:** ['Kiwi', 'Orange', 'banana', 'cherry']

**Case Insensitive Sort:** we can use built-in functions as key functions when sorting a list. So if you want a case-insensitive sort function, use str.lower as a key function.

**Example – 1:**

List1 = ["banana", "Orange", "Kiwi", "cherry"]

List1.sort(key = str.lower)

print(List1)

Output: ['banana', 'cherry', 'kiwi', 'orange']